

## **APPENDIX 5**

```
/**
 * IPS snmpwalk functions
 * @file ips_snmpwalk.h
 * @author jmccaskey
 */

#ifndef SNMPWALK__H
#define SNMPWALK__H

/**
 * Function to walk a specified device and oid. The function should be passed an
 * ips_device structure for the device to walk, a string for the oid to walk
 * a string to match against the returned values for each oid in the walk, and
 * a queue structure in which it will place a series of index_nodes. The optional
 * stop_after_one argument can be used to tell the function to quit after one match.
 */
int ips_snmpwalk(struct ips_device *device, char *type, char *start_oid, unsigned char *match_string,
queue *index_queue, int stop_after_one);

#include "snmpwalk.c"

#endif
```

```

/**
 * IPS snmpwalk functions
 * @file snmpwalk.c
 * @author jmccaskey
 */

int ips_snmpwalk(struct ips_device *device, char *type, char *start_oid, unsigned char *match_string,
queue *index_queue, int stop_after_one) {
    //setup some net-snmp stuff
    void *ss;
    struct snmp_session session, *sptr;
    struct snmp_pdu *pdu;
    struct snmp_pdu *response;
    oid root[MAX_OID_LEN];
    size_t root_length = MAX_OID_LEN;
    oid name[MAX_OID_LEN];
    size_t name_length = MAX_OID_LEN;
    struct variable_list *vars;
    int status, liberr, syserr;
    char *errstr;
    int count=1;

    //setup snmp session options
    session = ips_snmp_sess_init(device);

    //open session
    ss = snmp_sess_open(&session);
    if(!ss) {
        /* Error codes found in open calling argument */
        snmp_error(&session, &liberr, &syserr, &errstr);
        flockfile(stdout);
        fprintf(stdout, "snmpwalk: %s\n", errstr);
        funlockfile(stdout);
        free(errstr);
        if (response)
            snmp_free_pdu(response);
        snmp_sess_close(ss);

        return(6);
    }

    sptr = snmp_sess_session(ss);

    //add oid to walk
    get_node(start_oid, root, &root_length);
    memmove(name, root, root_length * sizeof(oid));
    name_length = root_length;
    int running = 1;
    while (running==1) {
        /** Create the pdu for the get next request and add the object name to it */
        pdu = snmp_pdu_create(SNMP_MSG_GETNEXT);
        snmp_add_null_var(pdu, name, name_length);

        /** Get the response from the device */
        status = snmp_sess_synch_response(ss, pdu, &response);
        if(status == STAT_SUCCESS) {
            if(response->errstat == SNMP_ERR_NOERROR) {

```

```

        for(vars = response->variables; vars; vars = vars->next_variable) {
            /** Check that the resulting variable is part of the desired tree */
            if((vars->name_length < root_length) || (memcmp(root, vars-
>name, root_length * sizeof(oid))!= 0)) {
                //not in the right sub tree
                running = 0;
                continue;
            }

            /** Print out variable */
            //print_variable(vars->name, vars->name_length, vars);

            /** Check if this is one of the values we are looking for... */
            char delim[1] = {'.'};
            char *oid_string;
            assert(oid_string=malloc(4000));
            int match = 0;
            //tunnel type specific matching code
            if(strcmp(type, "cisco_ipsec")==0) {
                //match each individual tunnel octet using the raw snmp
                if(vars->val.bitstring[0]==match_string[0] && vars-
                && vars->val.bitstring[2]==match_string[2] && vars-
                && vars->val.bitstring[3]==match_string[3]) {
                    match = 1;
                }
            } else if(strcmp(type, "netscreen_ipsec")==0) {
                char value[100];
                snprint_value(value, 100, vars->name, vars-
                && vars->val.bitstring[0]==match_string[0] && vars-
                && vars->val.bitstring[2]==match_string[2] && vars-
                && vars->val.bitstring[3]==match_string[3]) {
                    match = 1;
                }
            } else if(strcmp(type, "altiga_ipsec")==0) {
                char value[100];
                snprint_value(value, 100, vars->name, vars-
                && vars->val.bitstring[0]==match_string[0] && vars-
                && vars->val.bitstring[2]==match_string[2] && vars-
                && vars->val.bitstring[3]==match_string[3]) {
                    match = 1;
                }
            } else if(strcmp(type, "ips_emulated")==0) {
                char value[100];
                snprint_value(value, 100, vars->name, vars->name_length, vars);
                if(strcmp(value, match_string)==0) {
                    match = 1;
                }
            } else {
                flockfile(stdout);
                fprintf(stdout, "unknown type: %s\n", type);
                funlockfile(stdout);
                //.....
            }

            //check if this was a match -- if so get the index and queue it up
            if(match == 1) {

```

```

//we have a match... figure out what the index was
snprint_objid(oid_string, 4000, vars->name, vars->name_length);
char *strindex = NULL;
char *temp = NULL;
char *pos;
pos = oid_string;

//strsep is a gnu c specific extension... it replaces the non thread safe (and
slower) strtok from ansi c...
temp = strsep(&pos, delim);
while(temp != NULL) {
    strindex = temp;
    strcpy(strindex, temp);
    temp = strsep(&pos, delim);
}

//push the index into the queue
index_node *inode;
assert(inode = malloc(sizeof(*inode)));
flockfile(stdout);
fprintf(stdout, "Value of index: %s\n", strindex);
funlockfile(stdout);
inode->value = atoi(strindex);
queue_put(index_queue, (queue_node *)inode);

if(stop_after_one==1) {
    //we are done if we only want one value to be found
    running = 0;
}

#ifdef DEBUG
flockfile(stdout);
fprintf(stdout, "Session Id Match: %d\n", inode->value);
funlockfile(stdout);
#endif

}

free(oid_string);

/** Check that the value isn't an exception (not at the end yet) */
if ((vars->type != SNMP_ENDOFMIBVIEW) &&
    (vars->type != SNMP_NOSUCHOBJECT) &&
    (vars->type != SNMP_NOSUCHINSTANCE)) {
    /** Check that the oids are increasing... no infinite loops!

*/

    if (snmp_oid_compare(name, name_length, vars->name, vars-
>name_length) >= 0) {

        flockfile(stdout);
        fprintf(stdout, "Error: OID not increasing: ");
        fprintf_objid(stdout, name, name_length);
        fprintf(stdout, " >= ");
        fprintf_objid(stdout, vars->name, vars->name_length);
        fprintf(stdout, "\n");
        funlockfile(stdout);
        running = 0;
        continue;
    }
}

```

```

        /** Move the new variable into the old one so we can
continue doing get next's */
        memmove((char *) name, (char *) vars->name, vars-
>name_length * sizeof(oid));
        name_length = vars->name_length;
    } else {
        /** An exception value was found, probably at the end of
the tree we are walking */
        running = 0;
        continue;
    }
} else {
    /** There was an error in the response */
    running = 0;
    if (response->errstat == SNMP_ERR_NOSUCHNAME) {
        flockfile(stdout);
        fprintf(stdout, "End of MIB\n");
        funlockfile(stdout);
    } else {
        flockfile(stdout);
        fprintf(stdout, "Error in packet.\nReason: %s\n",
snmp_errstring(response->errstat));
        funlockfile(stdout);
        if (response->errindex != 0) {
            flockfile(stdout);
            fprintf(stderr, "Failed object: ");
            for (count = 1, vars = response->variables; vars && count !=
response->errindex;
                vars = vars->next_variable, count++)
                /*EMPTY*/;
            if (vars)
                fprintf_objid(stdout, vars->name, vars-
>name_length);
            fprintf(stdout, "\n");
            funlockfile(stdout);
        }
    }
} else if (status == STAT_TIMEOUT) {
    flockfile(stdout);
    fprintf(stdout, "Timeout: No Response from.%s\n", session.peername);
    funlockfile(stdout);
    running = 0;
} else {
    snmp_sess_perror("snmpwalk", ss);
    running = 0;
}
if (response)
    snmp_free_pdu(response);
}
snmp_sess_close(ss);

return(0);
}

```